Interactive Stereoscopic Visualization of Large-Scale Astrophysical Simulations

Ralf Kaehler[a] and Tom Abel[a]

[a]KIPAC/SLAC, 2575 Sand Hill Road, Menlo Park, USA

ABSTRACT

In the last decades three-dimensional, time-dependent numerical simulations have become a standard tool in astrophysics and cosmology. This gave rise to a growing demand for analysis methods that are tailored to this type of simulation data, for example high-quality visualization approaches such as direct volume rendering and the display of stream lines. The modelled phenomena in numerical astrophysics usually involve complex spatial and temporal structures, and stereoscopic display techniques have proven to be particularly beneficial to clarify the spatial relationships of the relevant features. In this paper we present a flexible software framework for interactive stereoscopic visualizations of large time-dependent, three-dimensional astrophysical and cosmological simulation datasets. It is designed to enable fast and intuitive creation of complete rendering workflows, from importing datasets, the definition of various parameters, including camera paths and stereoscopic settings, to the storage of the final images in various output formats. It leverages the power of modern graphics processing units (GPUs) and supports high-quality floating-point precision throughout the whole rendering pipeline. All functionality is scriptable through Javascript. We give several application examples, including sequences produced for a number of planetarium shows.

Keywords: Scientific Visualization, Astrophysics, Stereoscopic Rendering, Direct Volume Rendering

1. INTRODUCTION

Because of rapidly increasing computational resources and algorithmic advances, numerical simulations in astrophysics and cosmology have become very sophisticated and now allow detailed ab initio studies of many different physical processes. Examples include the evolution of the large-scale structure in the Universe from initial density perturbations generated shortly after the Big Bang, the generation of the first stars and galaxies from primordial gas clouds as well as the so-called reionization era of the cosmos.

Two types of numerical techniques are particularly popular in numerical astrophysics. N-body and smoothed-particle-hydrodynamics (SPH) codes generate large unstructured point sets to store information associated with their computational elements, e.g. dark matter and star particles. Grid-based hydrodynamics simulations often employ cubical grid cells to discretize the computational domain; locally refined structured adaptive grids (AMR) are widely-used in this context, because they allow the codes to cover many orders of spatial and temporal scales. Visualizing the results of these simulations is challenging, because of the sheer data sizes as well as the sophisticated data structures. This paper presents a software framework for interactive stereoscopic visualization of this kind of simulations on standard desktop workstations. In order to achieve interactive rendering performance even for larger datasets, the software leverages state-of-the-art programmable graphics hardware which gives the user full flexibility to tailor the rendering pipeline to a wide range of application scenarios. It supports various popular 3D visualization methods for scalar and vector-valued fields, like *GPU-assisted ray-casting* and the display of illuminated streamlines and enables the user to define workflows in an intuitive fashion, starting with the import of the datasets into the application to the export of the final rendering results in full floating-point precision. It has been successfully applied to generate sequences for various television and planetarium shows, including *"Journey to the Stars"*, produced by the *American Museum of Natural History*[1] and *"Life: A Cosmic Story"*, produced by the *California Academy of Sciences*.[2]

Further author information: (Send correspondence to R. Kaehler)
R. Kaehler: E-mail: kaehler@slac.stanford.edu, Telephone: 1 650 926 2884
T. Abel: E-mail: tabel@slac.stanford.edu, Telephone: 1 650 926 2421

Stereoscopic Displays and Applications XXIII, edited by Andrew J. Woods, Nicolas S. Holliman, Gregg E. Favalora, Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 8288, 82882O
 · doi: 10.1117/12.909258

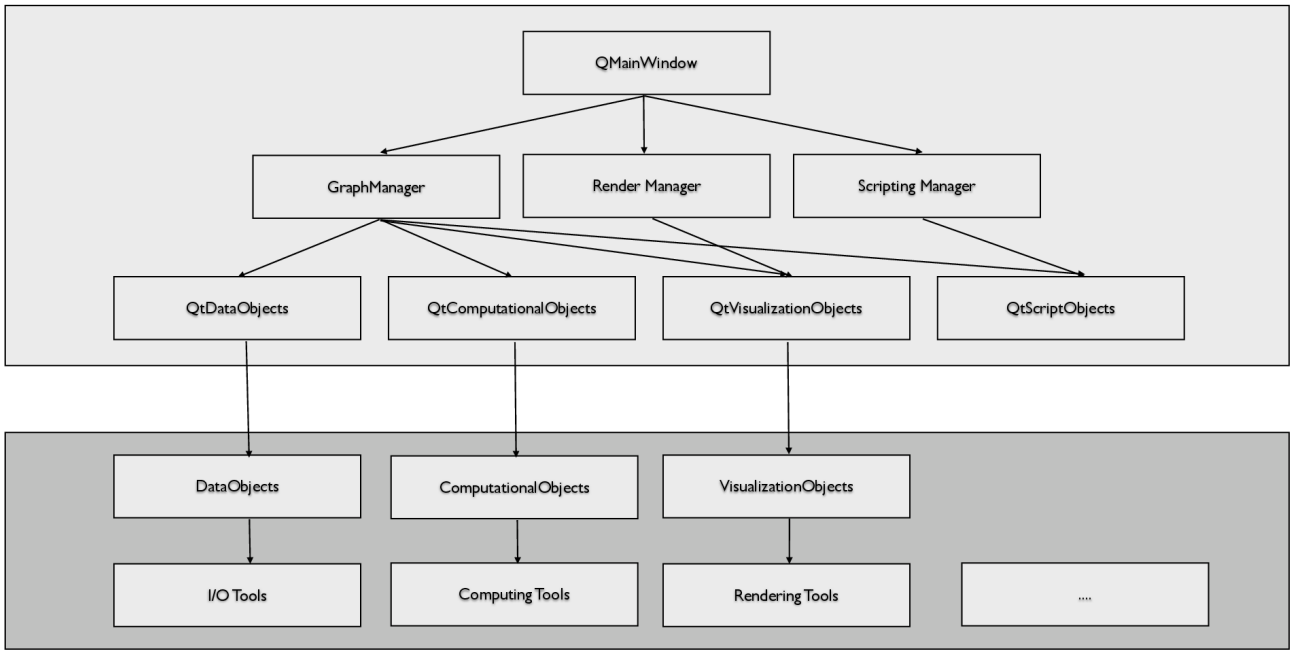

Figure 1. The software is divided into two layers: a *GUI*-independent library that contains classes that implement the various rendering algorithms, the data model and I/O routines and *Qt*-dependent classes that provide the *GUI*-front-end to this library.

The remainder of this contribution is organized as follows. In the next Section 2 we will give an overview of the design and architecture of the software. We will then explain how to establish a typical rendering workflow in Section 2.1 and end with a discussion of several rendering projects we have completed with it in Section 3.

2. DESIGN AND ARCHITECTURE

The design goal was to create a light-weight 3D visualization framework for widely-used data formats in numerical astrophysics and cosmology. It should allow interactive rendering performance on standard desktop computers by leveraging the rapidly increasing power of commodity graphics hardware and enable the user to define specific workflows for various application scenarios in a flexible and intuitive way via a graphical user interface (*GUI*).

We chose *C++* as the primary programming language, the *Qt* framework[3] for the graphical user interface and *OpenGL-API*[4] for rendering. A simplified class layout is shown in Figure 2. The software is organized into two layers. A *GUI*-independent library that contains classes that implement the various rendering and visualization algorithms, as well as the data model and IO routines. The second layer consists of *Qt*-dependent classes that provide the GUI to the library classes. Library classes fall into three different categories: *data classes*, *computational classes* and *rendering classes*. *Data classes* represent datasets, which are either stored on local disks, accessed over the network or generated on-the-fly during runtime. The second type, *computational classes* operate on *data classes* and either modify the data directly or use them to derive new datasets. Examples are the generation of gradients from scalar input data and the extraction of streamlines from vector-valued data fields. *Rendering classes* implement visualization routines, either by directly calling the *OpenGL-API* or by utilizing library classes that encapsulate certain *OpenGL* concepts, like *framebuffer objects*, *textures*, *graphic shaders*.

The *GUI* is divided into four separate windows that can be combined via *Qt*'s docking mode, see Figure 2. One of them is used to define the overall workflow. It represents instances of the *data classes*, *computational classes* and *rendering classes* as separate nodes in a workflow graph, as shown on the top-left of Figure 2. The user can add new nodes by selected them from the application's main menu and interact with the graph to

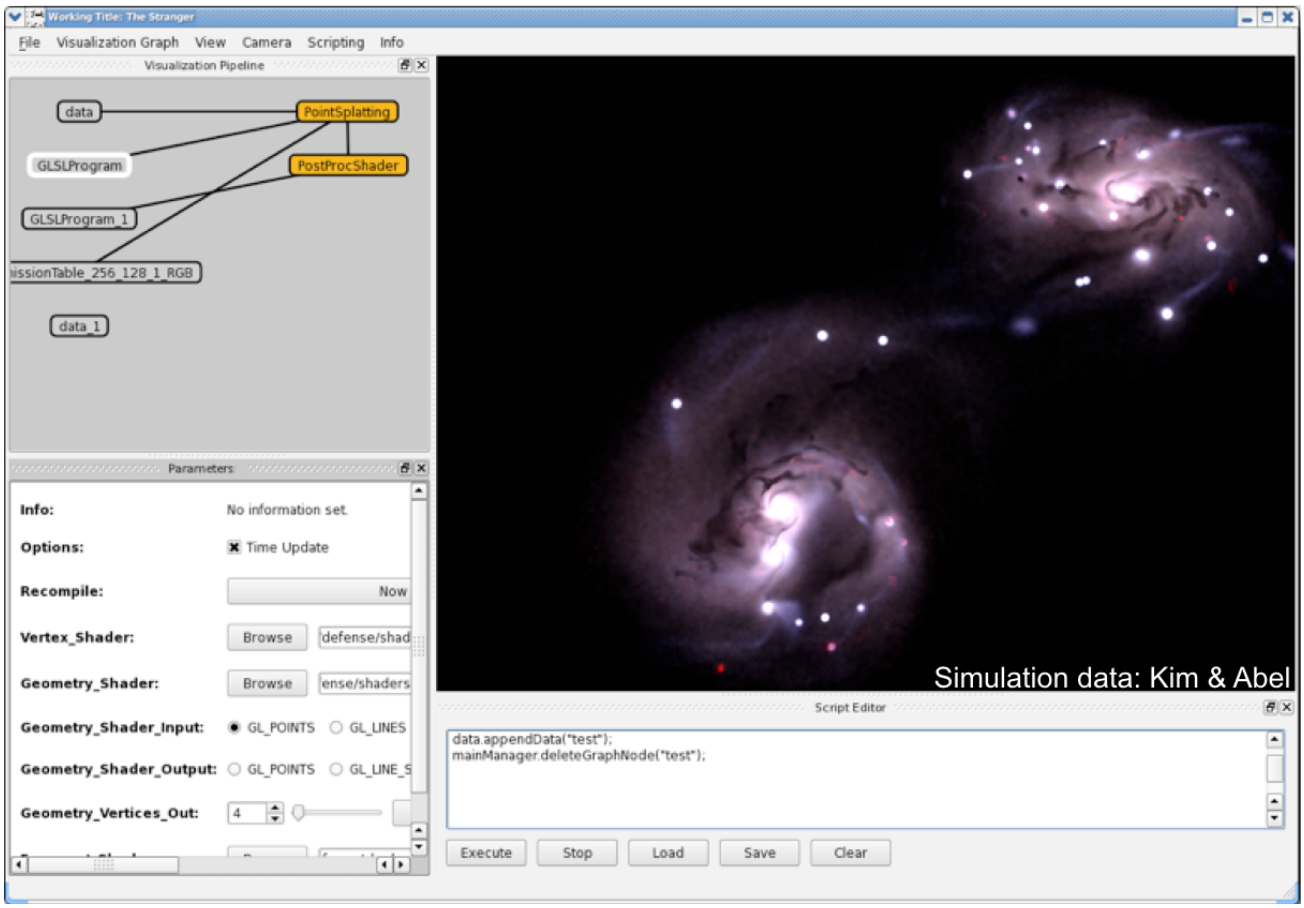

Figure 2. The graphical interface of the application is divided into four separate subwindows that allow the user to define the overall workflow (top-left), adjust rendering parameters (bottom-left), write scripting code (bottom-right) and interact with the final rendering results (top-right).

establish connections between the objects. If a node is selected, several *GUI* widgets are displayed in a separate subwindow which allows the user to adjust the node-specific paramaters.

The third subwindow displays the rendering results. It is derived from the *QGLWidget* base class and provides an *OpenGL* rendering context. It stores a list of all currently active rendering objects and calls their rendering methods in the order of assigned priority ids, which can be changed in the node's parameter *GUI*. The user interacts with this window via the mouse to change the camera parameters, like positon and zoom level. The partial results of the rendering nodes are combined in offscreen *RGBA floating-point buffers* and the final result is displayed in the framebuffer. The whole workflow, including the camera information and of the node specific parameters can be saved to disk and reloaded to restore the session. Communication between different objects is established via *Qt*'s *signals & slots* technique, where *signals* are emitted for certain events that require re-computations, respectively re-renderings, for example changes of the camera location or modifications of *data objects*. In the following we will clarify these concepts by describing how a typical workflow is established.

2.1 Workflow

Typically the first step to define a new workflow is to select one or more input datasets via a file dialog. Currently scalar- and vector valued data defined on unstructured points, uniform as well as structured adaptive mesh refinement (AMR) grids are supported. Predefined readers exists for various file formats, including those used by the popular simulation packages Enzo,[5] RAMSES[6] and Gadget[7] . It is easy to add new, customized readers by deriving from an abstract base class and implementing a small number of virtual functions. Once a

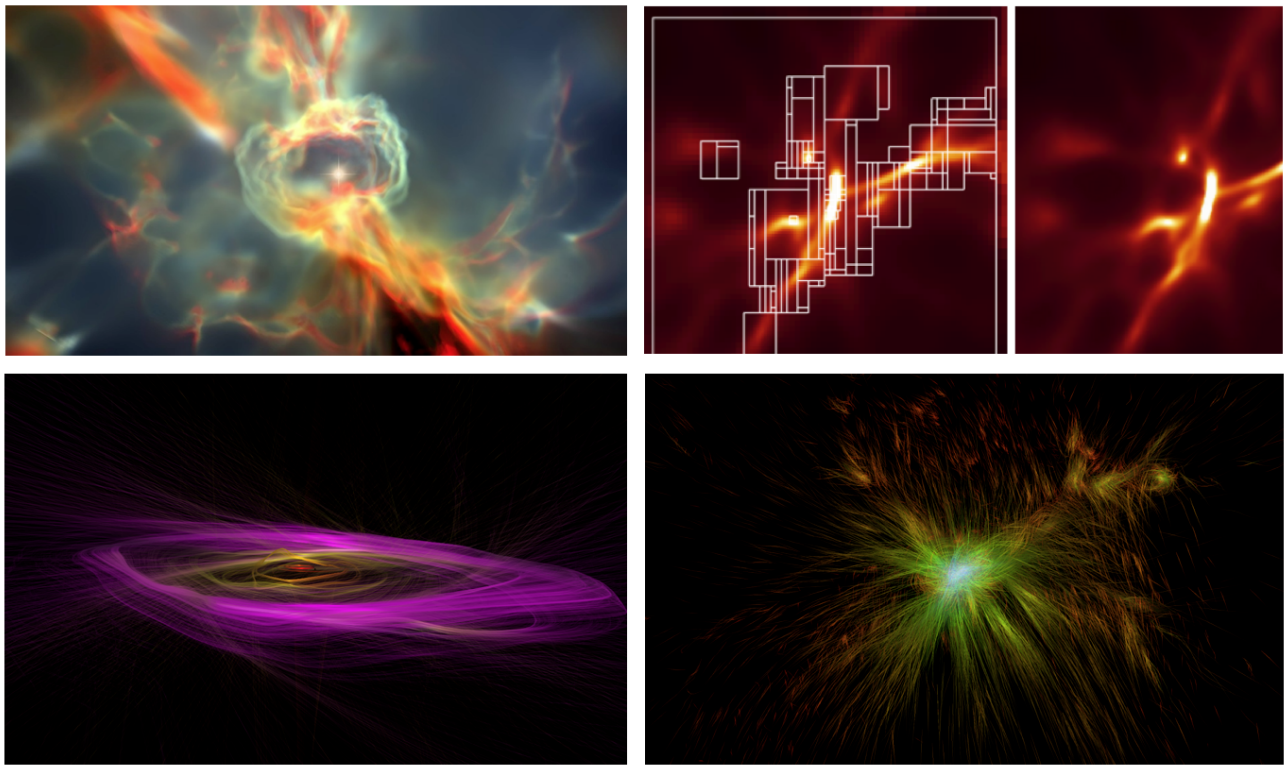

Figure 3. Examples of different visualization methods supported by the framework: GPU-assisted raycasting (top-left), extraction of planar slices (top-right), illuminated streamlines (bottom-left) and particle pathlines (bottom-right).

dataset is imported, is is represented as a node in the workflow graph. By selecting the node, information about the dataset is displayed in a separate window. Examples are the names of the data fields, the number of particles for point-based data and the number of refinement levels for AMR data. Initially only the meta data of each dataset is loaded, whereas the access to the data itself is deferred until a computational or visualization routine actually requests it. In particular we support view-dependent access to AMR data, where the distance to the camera defines the resolution that is sufficient for different parts of the data domain.

As a next step the user typically selects computational and/or visualization objects from the application's main menu and connects the resulting nodes in the workflow graph to the input data objects. Selecting these nodes allows the users to modify the associated parameters in a separate window. The visualization objects use graphics *shaders* that are written in the *OpenGL Shading Language* and executed on the graphics hardware in a massively parallel fashion to define how the rendering primitives are processes in the graphics pipeline. The shaders can be modified and recompiled during runtime of the application, which gives the user a lot of flexibility to customize the visualization routines to application specific needs. One rendering approach that makes extensive use of graphics shaders is *GPU-assisted* raycasting,[8–10] see top-left of Figure 3. Here the data volume is sampled along the line of sight for each pixel and the accumulated colors and opacities are computed in separate instances of a *fragment shader*. By directly modifying the shader source code on-the-fly, the user can for example flexibly change the way different input datasets are combined to generate the final color intensities. It would be impossible to achieve this level of flexibility with hardcoded *GUI*-elements that only allow to change a relatively small number of parameters. Another example is the display of *illuminated streamlines*,[11] where the color of the line-segments is derived from the local vector field values inside a *vertex shader*, see Figure 3.

Partial results of rendering objects are combined by specifying the *OpenGL blending* and *depth-testing* state for each object individually along with the order in which the objects are processed. This allows the user for example to accumulate several semi-transparent rendering passes and realize various advanced computer graphics techniques like depth-peeling. Stereoscopic rendering is supported via *OpenGL*'s quad-buffered stereo mode with

an asymmetric viewing-frustum. Parameters like the eye-separation and the near and far-clipping planes can be adjusted in the *GUI* or via the application's scripting interface. The process of generating animations usually involves the generation of camera paths. We provide a camera path editor that employs a small number of user-defined key-frames to generates intermediate positions from these using cubic-spline interpolation. We support 32bit floating-point precision throughout the whole rendering pipeline by transferring data to the GPU via floating-point *textures* or *vertex buffers*, using render targets with 32bit per color-channel and exporting the rendering results via the *EXR* image format.

3. EXAMPLES

In this section we present various stereoscopic animations that were produced with our software framework. Figure 4 was rendered from a simulation of the re-ionization era in the early Universe, showing how the first galaxies in the Universe ionized the intergalactic gaseous material. The timespan covered by the simulation is from 500 million to about 1 billion years after the Big Bang and the spatial extend is 250 million light years across. Cold, neutral hydrogen regions are depicted by dark, opaque colors, while ionized material is rendered in blue and red colors. Temperature and density fields were combined to compute the final colors and opacities along the line of sight.

Figure 5 is a snapshot of a simulation that models the formation of the first stars in the Universe and how they interacted with their gaseous environment. The time range of the simulation is from 10 million to 250 million years after Big Bang. The camera is located inside a proto-galaxy that is several 1,000 light years across. A metal-free star formed in the central region. It is very heavy, around 30-100 solar masses and emits high-energy UV radiation, million times more than sun, thus quickly ionizing the surrounding hydrogen gas, that is shown in red. The radiation transport through the gaseous material was rendered using the GPU-assisted raycasting approach mentioned in Section 2.1. 2D versions of this animations have been used for the planetarium shows *"Journey to the Stars"*, produced by the *American Museum of Natural History*[1] and *"Life: A Cosmic Story"*, produced by the *California Academy of Sciences*.[2]

Figure 6 shows the formation of large-scale structures and massive galaxies in the early Universe, less than 3 billion years after the Big Bang. The simulation models the complicated interplay between the three main components of the galaxies, i. e. dark matter, stars and the interstellar and intergalactic gas. The computation was carried out on adaptive mesh refinement (AMR) grids with about 60 million cells and resolved the dark matter distribution and the stars within the galaxies with additional 40 million particles using about 40,000 time steps. The time span shown in this animation is 2 billion years and the region in the foreground has an extension of 50 million light years across. The animation focusses on galaxy collisions and mergers in the foreground. The color-coding is based on the age of the stars. Red colors depict massive star forming regions, blue and white colors indicate young bright stars, whereas yellow and brown colors represent older stars. The blueish regions in the background represent the large-scale distribution of hydrogen gas on a scale of about 300 million light years. A 2D version of this animation was used for the show *"The Big Bang"*, produced and shown at the *Hayden Planetarium* at the *American Museum of Natural History* in New York.

Figure 7 shows a visualization of a dark matter N-body simulation. 40 million particles represent a dark matter distribution is a region about 120 million light years across. The simulation starts from dark matter density fluctuations about 300 million years after the Big Bang and evolves them for more than 13 billion years. About 1000 gravitationally bound regions, so-called dark matter halos, are visible as bright yellow and white regions, each of them hosting a separate galaxy.

4. CONCLUSIONS

We presented a visualization framework that enables the user to intuitively and efficiently create complete rendering workflows for high-quality stereoscopic visualizations and animations of large three-dimensional datasets at interactive frame-rates. It leverages the capabilities of modern graphics processing units (GPUs) and supports full floating-point precision throughout the whole rendering pipeline. All functionality is scriptable through Javascript. We gave several application examples, including sequences that have been produced for a number of planetarium shows.

In the future we plan to run the software on GPU clusters, for example to enable the visualization of even larger datasets on high-resolution tiled-display walls. We also plan to extend the framework by customizable rendering routines written in *CUDA* and *OpenCL*.

5. ACKNOWLEDGMENTS

This work was partly support by the *National Science foundation* through award number AST- 0808398 and the LDRD program at the SLAC National Accelerator Laboratory as well as the Terman fellowship at Stanford University.

REFERENCES

[1] `http://www.amnh.org/rose/spaceshow/journey/`.

[2] `http://www.calacademy.org/academy/exhibits/planetarium/life/`.

[3] Blanchette, J. and Summerfield, M., [*C++ gui programming with qt 4, second edition*], Prentice Hall Press, Upper Saddle River, NJ, USA, second ed. (2008).

[4] Shreiner, D., [*OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*], Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd ed. (1999).

[5] OShea, B. W., Bryan, G., Bordner, J., Norman, M. L., Abel, T., Harkness, R., and Kritsuk, A., "Introducing Enzo, an AMR Cosmology Application," *ArXiv Astrophysics e-prints* (Mar. 2004).

[6] Fromang, S., Hennebelle, P., and Teyssier, R., [*RAMSES-MHD: an AMR Godunov code for astrophysical applications*], 743 (2005).

[7] Springel, V., "The cosmological simulation code gadget-2," *Monthly Notices of the Royal Astronomical Society* **364**(4), 31 (2005).

[8] Stegmaier, S., Strengert, M., Klein, T., and Ertl, T., "A simple and flexible Volume Rendering Framework for Graphics-Hardware-based Raycasting," in [*Fourth International Workshop on Volume Graphics*], 187–241 (2005).

[9] Kaehler, R., Wise, J., Abel, T., and Hege, H.-C., "GPU-Assisted Raycasting for Cosmological Adaptive Mesh Refinement Simulations," in [*International Workshop on Volume Graphics 2006*], 103–110, Eurographics / IEEE VGTC 2006, Boston (2006).

[10] Kaehler, R., Abel, T., and Hege, H.-C., "Simultaneous GPU-Assisted Raycasting of Unstructured Point Sets and Volumetric Grid Data ," *Eurographics/IEEE VGTC Symposium on Volume Graphics* **1**, 49–56 (2007).

[11] Zöckler, M., Stalling, D., and Hege, H.-C., "Interactive visualization of 3d-vector fields using illuminated stream lines," in [*Proceedings of the 7th conference on Visualization '96*], *VIS '96*, 107–ff., IEEE Computer Society Press, Los Alamitos, CA, USA (1996).

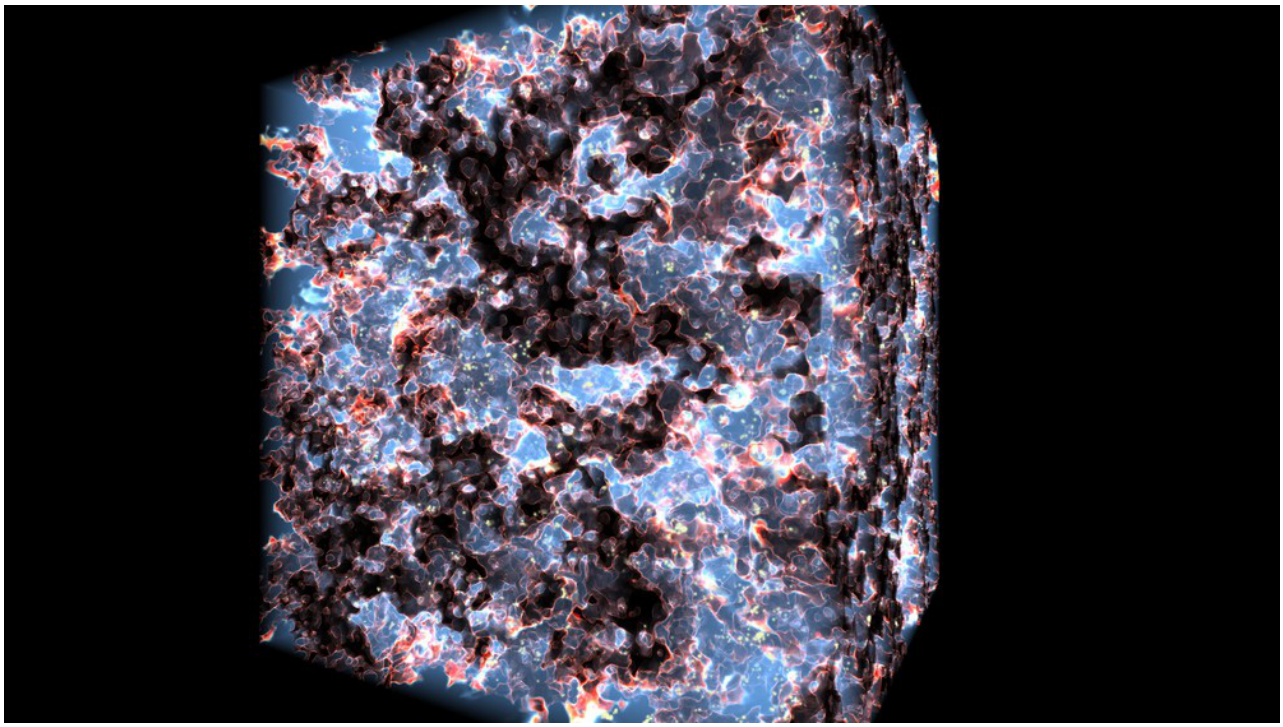

Figure 4. Video 1: A a simulation of the re-ionization era in the early Universe. Numerical simulation by Marcelo Alvarez (CITA) and Tom Abel. http://dx.doi.org/10.1117/12.909258.1

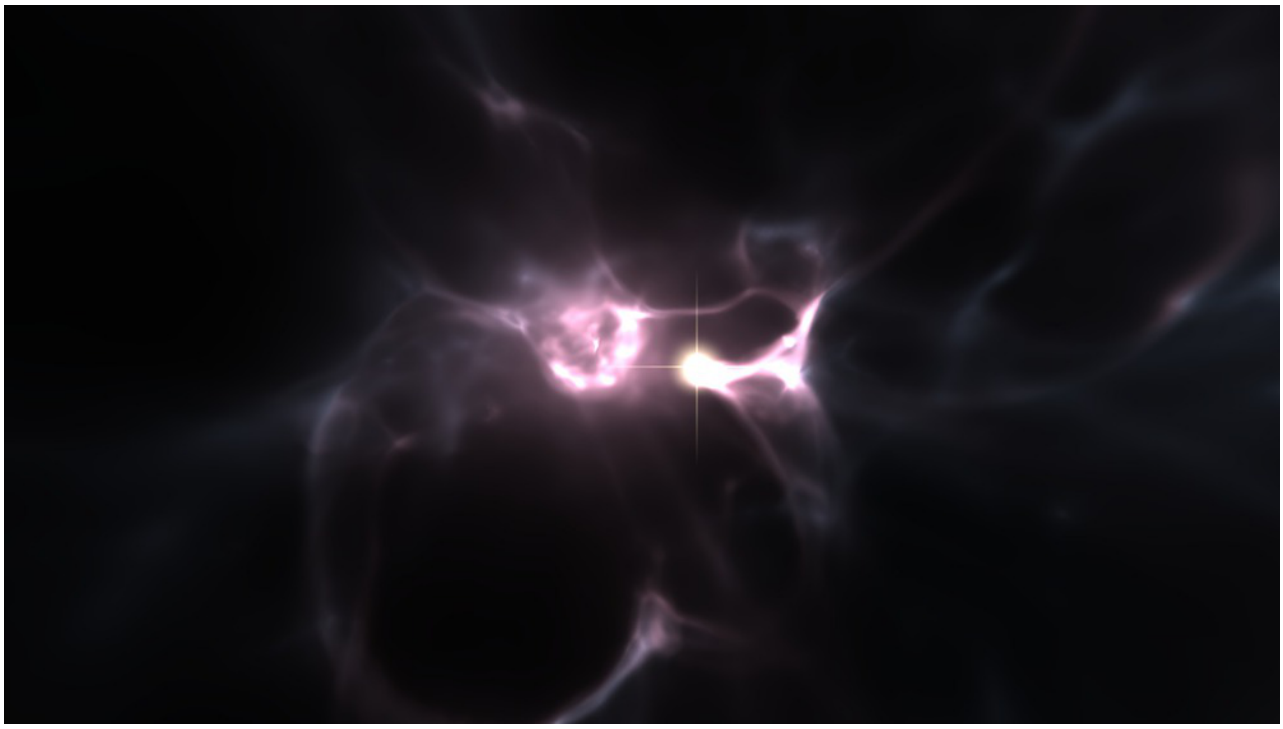

Figure 5. Video 2: A simulation of the formation of the first stars in the Universe and their feedback on their gaseous environment. Numerical simulation by John Wise (Georgia Tech) and Tom Abel (KIPAC). http://dx.doi.org/10.1117/12.909258.2

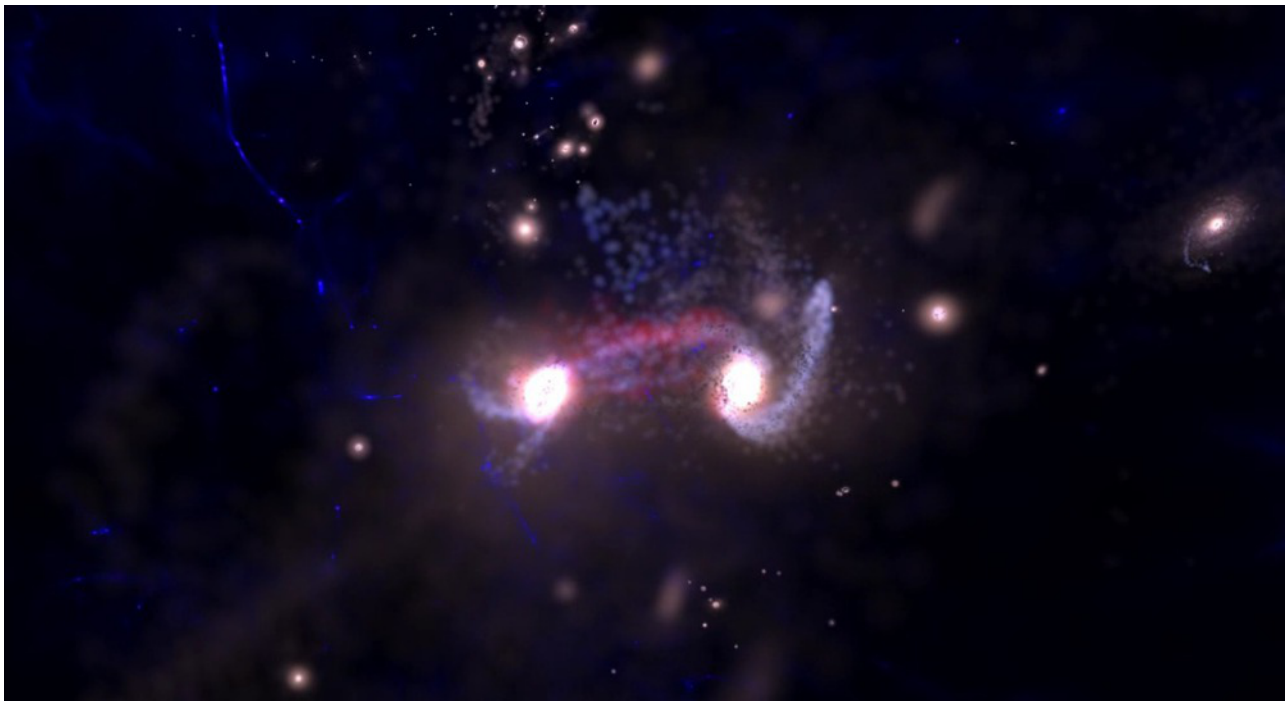

Figure 6. Video 3: This visualization shows the formation of large-scale structures and massive galaxies in the early Universe, less than 3 billion years after the Big Bang. Numerical simulation by Ji-hoon Kim (UCSC) and Tom Abel. http://dx.doi.org/10.1117/12.909258.3

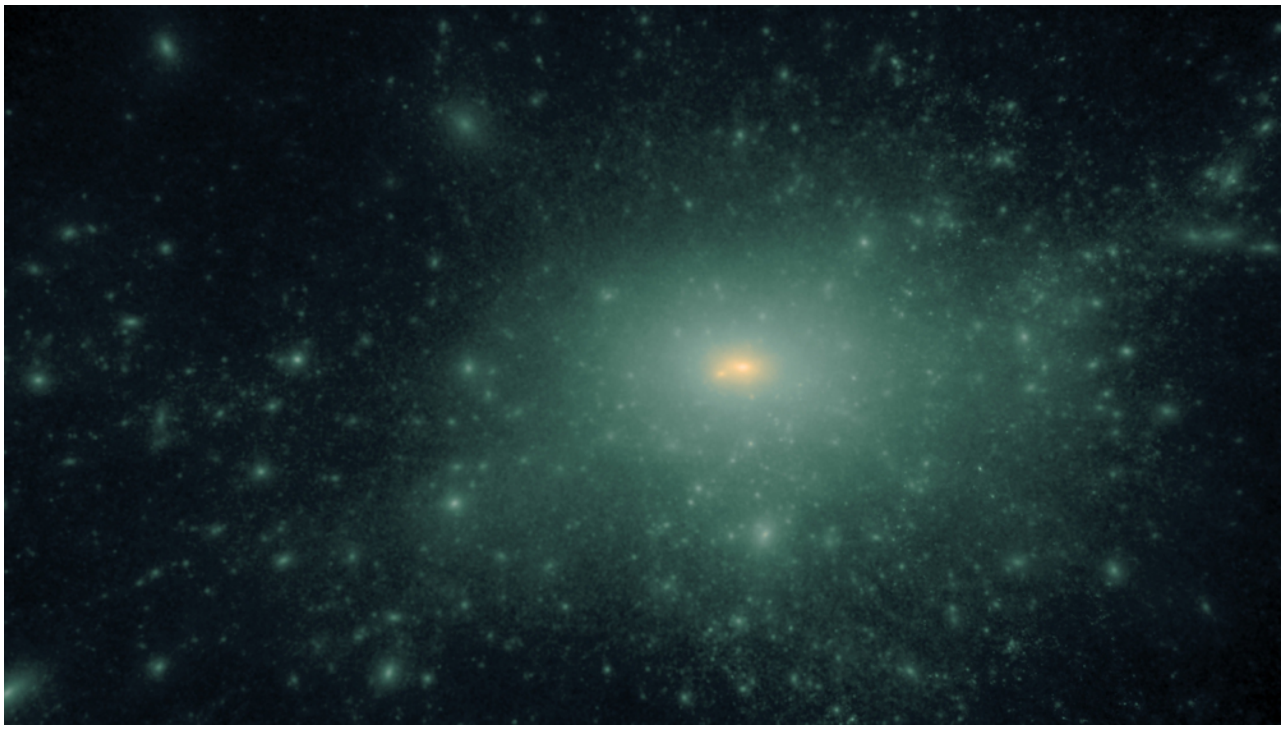

Figure 7. Video 4: Simulation of the large-scale structure formation in the Universe using 40 million dark matter particles. Simulation by Heidi Wu, Oliver Hahn and Risa Wechsler (Stanford University).http://dx.doi.org/10.1117/12.909258.4